Zewail City of Science and Technology

Big Data Analytics
CIE 427

Mini-Project 2 Technical Report

Omar Elsakka	201900773
Omar Emad	201901607

June 19, 2023

Contents

Our Code is provided here Colab Notebook

1 Introduction

League of Legends (LOL) is an online battle game developed by Riot Games. This game is played with two teams. Each team tries to defeat the other team using strategic decisions and variant fancy tactics. Players control their matches using champions (there are a set of champions to choose from). The main aim is destroying the enemy, gathering items, and getting key positions as much as it could. It has become one of the most popular video games in the world, with over 100 million Players.

2 Collecting Data Optimization

It is supposed to collect more than 75K match data to do our analysis on them. But here is one challenge, the API key of League of legends is deactivated every 24 hours. Moreover, the API gets 100 records every 2 minutes. So, it is not applicable to save 75K matches, it would take $\frac{75000}{100}.2 = 1500$ minutes $= 25$ hours multiply this by 4 because every match needs 4 requests and we end up with 100 hours. So, We used around 7 APIs to collect this data, but the server was smarter than we thought. Our new accounts were blacklisted. So, we tried the same idea but after waiting 24 hours so that our new accounts are no longer blacklisted. We managed to collect 100K Matches in less than 24 hours.

Another main flaw in the Riot api is that it does not allow its users to access match data with a single request.

to get match data we first need to scrape the Ranked divisions for Summoner IDs, Those summoner IDs then are user to collect what Riot calls PUUID this PUUID is then used to scrape the player's match history for match IDs. Here we finally can start Collecting match data with the gatherd match IDs.

3 Scrapped Data

We scrapped data into 5 files:

- ChampClass.txt: Contains classes of each champion
- Id_To_ItemName.txt: Contains the ID for each Item.
- Id_To_ChampionName.txt: Contains the ID for each champion.
- Data.csv: Contains the match data, as it contains the champions ID who win the match and their item IDs, and the same with the team who loses the match, duration time in Minutes. Also, the bans ,meta for both teams and game version at which the game occurred.

Notes:

- Champions are the characters that players use in game.
- Item is just accessories for champions that may help in success.

after collecting data we used the game-Batch Column to partition the data into batches for the bonus.

4 Cleaning Data

most of the cleaning was done while scrapping the data:

- selecting the data we want such as champion and items, bans, some meta data like dragon kills.
- Converting match duration to be in Minutes.
- Removing Redundant Items.

while working with the data we noticed that there are duplicates and that some matches had their game duration in milliseconds. so we had to divide the game duration by 1000 in case it was more some limit.

- Dropping duplicated data around 5K matches.
- Converting match duration to be in minutes.

5 Data Analysis and its Code Design

5.1 Analysis related to Champions

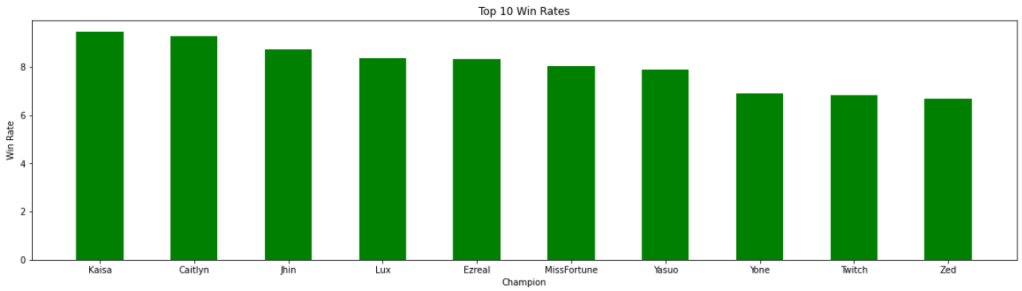
We want to see the win rate, lose rate, Pick rate, Ban rate ordered in Descending way, Champion synergies, or duos. Those visualizations could be done using basic functions in pyspark like map, flatmap, reduceByKey, filter and sortBy. For example, for getting the WinRate

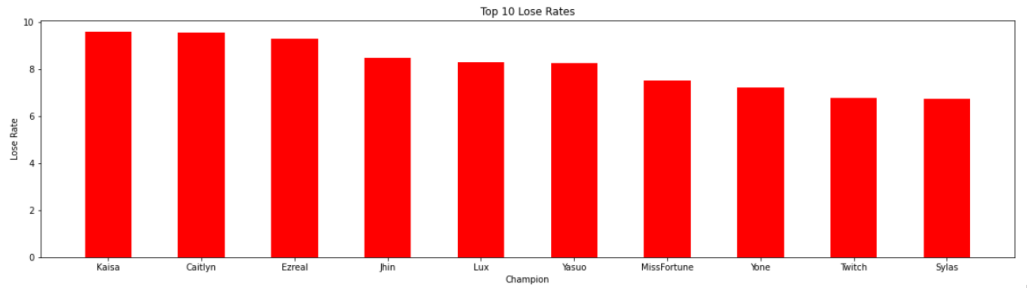
```
ChampionsWon=df.select(["Champ1Team1","Champ2Team1","Champ3Team1","Champ4Team1",
"Champ5Team1"])

WinRates = ChampionsWon.rdd.flatMap(lambda x:x)\
           .map(lambda champ:(champ.split(',')[0],1)) \
           .reduceByKey(lambda c1,c2:c1+c2) \
           .map(lambda champ:(champ[0],round((champ[1]/(df_count))*100,2)))\
           .sortBy(lambda champ: champ[1], ascending=False)\
           .map(lambda champ:(Id_to_ChampName.get(champ[0]),champ[1])) \
           .collect()
```

The Team1 column contains winning teams per match. So, what is done in the above code is just counting the number of wins per each champion and calculating it as a percentage then map it each Champion ID to its name.
For the losing rate, We were the same idea but using the losing team (Team 2).
Also, for ban rate and pick rate, it is the same as before in win rate but using Ban Data and Played champions Data instead of Winner Data.

```
ChampionBans=df.select(['BansTeam1','BansTeam2'])

Champions=df.select(["Champ1Team1","Champ2Team1","Champ3Team1","Champ4Team1",
"Champ5Team1","Champ1Team2","Champ2Team2","Champ3Team2","Champ4Team2","Champ5Team2"])
```

Here is the Win Rate and Lose Rate graphs





Now, for Champion Synergies and duos, We did a function to get champion combinations in winning or losing

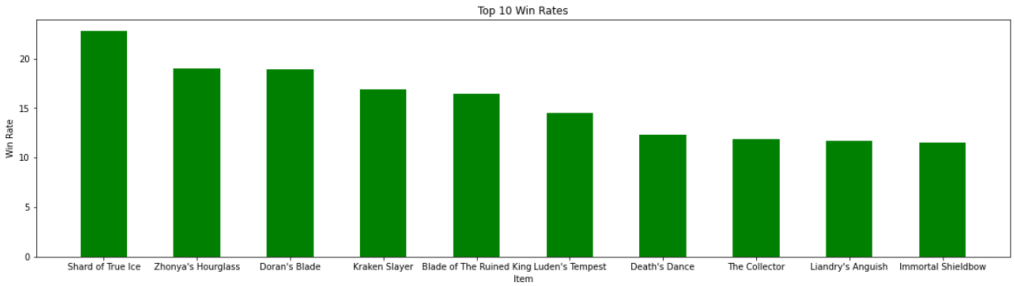
```
def GetChampCombinations(string):
    champs = string.split('_')
    #get the id of all the champions in the team
    ChampIds = [int(champ.split(',')[0]) for champ in champs]
    #replace the id with the name
    ChampList = [Id_to_ChampName.get(str(id)) for id in ChampIds]
    #sort the names to avoid tuples that contain the same two champions but in different o
    ChampList.sort()
    #return the combinations of the given team
    return list(combinations(ChampList, 2))

SynergiesWon = ChampionsWon.rdd.flatMap(lambda x:x)\
            .map(lambda champList:GetChampCombinations(champList)) \
            .flatMap(lambda pair:pair) \
            .map(lambda pair:(pair,2)) \
            .reduceByKey(lambda pair1,pair2:pair1+pair2)
```

The same idea for Synergies Lose but using ChampionsLose instead of ChampionsWon Data.

5.2 Analysis related to Items

For the Winning rate, Losing rate, and Pick rate of Items, we did the same as above by replacing Champions Data with Items Data. For example, for Winning rate:

```
ItemWinRates= ChampionsWon.rdd.flatMap(lambda x:x)\
            .map(lambda Item:(Item.split(','))) \
            .flatMap(lambda Item:Item[1:]) \
            .map(lambda Item:(Item,1)) \
            .reduceByKey(lambda Item1,Item2:Item1+Item2) \
            .map(lambda Item:(Item[0],round((Item[1]/(df_count))*100,2))) \
            .filter(lambda Item: int(Item[0])>0 and Item[0] not in RedundantItems) \
            .map(lambda Item:(Id_to_ItemName.get(Item[0]),Item[1])) \
            .filter(lambda Item: Item[0] != None) \
            .sortBy(lambda Item: Item[1], ascending=False) \
            .collect()
```



for Item synergies (Item with Champion). We just map each champion to its picked items, then sort them by items. We did it for the item picked in the first place, items picked in the second place, and items picked in the third place. The same idea with (Item with Class)

```
ItemSynergies = Champions.rdd.flatMap(lambda x:x)\
            .map(lambda champ:(champ.split(','))) \
            .flatMap(lambda champ:[((champ[0],item),1) \
            for item in champ[1:] if int(item)>0 and item not
            in RedundantItems and item in OurChosenItems]) \
            .reduceByKey(lambda champItem1,champItem2:champItem1+champItem2) \
```

```
.map(lambda Item:((Id_to_ChampName.get(Item[0][0]),
Id_to_ItemName.get(Item[0][1])),Item[1])) \
.filter(lambda Item: Item[0][0] != None) \
.sortBy(lambda Item: Item[1], ascending=False) \
.collect()
```

5.2.1 Item suggestions

Here we developed a function that returns back the most assigned items for each element in the entered list. For example, if the entered list is (OurChosenChamps=['Zed', 'Zac']).

And we made a colab notebook that gamers can use to get real-time recommendations by just selecting the champion Notebook

```
Zed
        Ionian Boots of Lucidity
        Eclipse
        Youmuu's Ghostblade
        Long Sword
        Serylda's Grudge
        Ravenous Hydra
**************************************
Zac
        Sunfire Aegis
        Ionian Boots of Lucidity
        Demonic Embrace
        Thornmail
        Plated Steelcaps
        Refillable Potion
**************************************
```

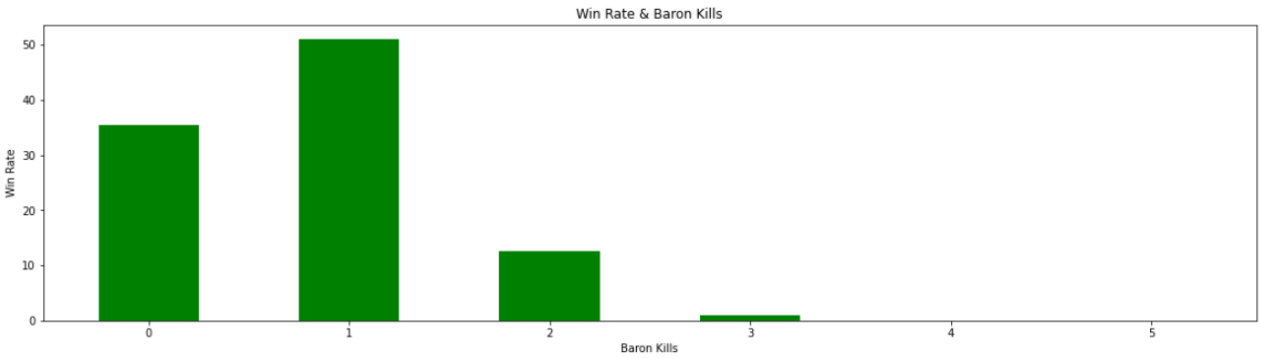
6 Proposal Questions and its code design

6.1 The association between Win rate and (Baron kills, Dragon kills, and Rift kills)

If we want to study the correlation between Win rate and Baron kill numbers we will just count the number of Baron kills in each winning team.

```
BaronWinRates = MetaWon.rdd.flatMap(lambda x:x)\
        .map(lambda champ:(champ.split(',')[1],1)) \
        .reduceByKey(lambda g1,g2:g1+g2) \
        .map(lambda game:(game[0],round((game[1]/(df_count))*100,2)))\
        .sortBy(lambda game: game[0])\
        .collect()
```

So, the result is high win rate is associated with 1 Baron kill in match.



The same is applied to Dragon kill numbers, Rift kill, and champion kill numbers.

In the case of studying the correlation between the Win rate with both Baron kills and Dragon kills.

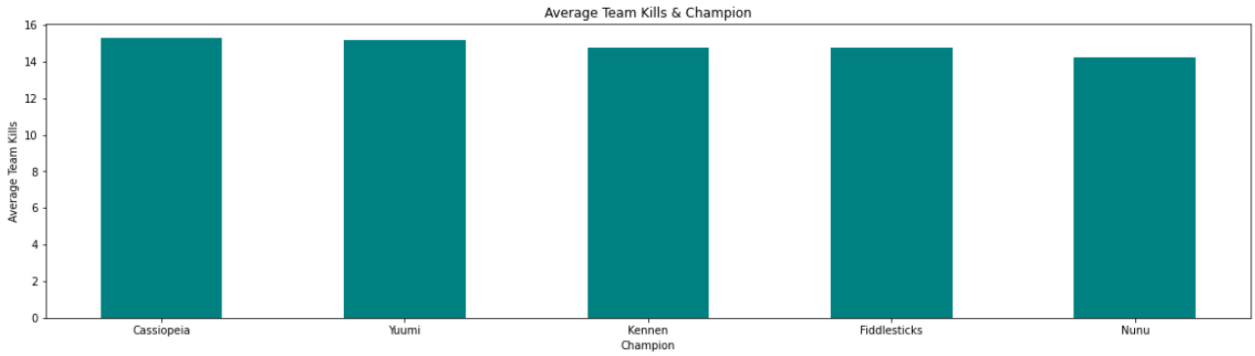
```
BaronDragonWinRates = MetaWon.rdd.flatMap(lambda x:x)\
            .map(lambda champ:(champ.split(',')[1]+"&"+champ.split(',')[2],1)) \
            .reduceByKey(lambda g1,g2:g1+g2) \
            .map(lambda game:(game[0],round((game[1]/(df_count))*100,2)))\
            .sortBy(lambda game: game[1],ascending=False)\
            .collect()
```

6.2 The association between (average team kills, average Tower kills, Game Time) and each champion

We first needed to include the meta data and gameduration inside each champion. So, we had to make a small modification in our dataset. the champion row now carries its items, team meta data and game duration.

To get the correlation between the average team kills and each champion, we can get the number of team kills and each champion in the team.

```
Champion_TeamKills = df_New.rdd.flatMap(lambda x:x)\
            .map(lambda champ:(champ.split(',')[0],(int(champ.split(',')[7]),1))) \
            .reduceByKey(lambda c1,c2:(c1[0]+c2[0],c1[1]+c2[1])) \
            .map(lambda champ:(champ[0],round((champ[1][0]/champ[1][1]),2)))\
            .sortBy(lambda game: game[1],ascending=False)\
            .map(lambda champ:(Id_to_ChampName.get(champ[0]),champ[1])) \
            .collect()
```



The same could be applied to average tower kills and average Game time instead of average team kills.

7 Machine Learning

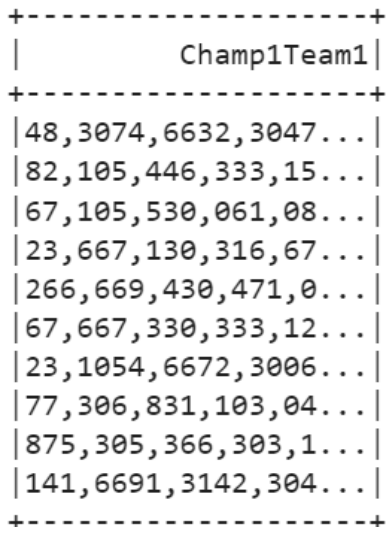
7.1 Challenges

In this task, We are trying to estimate the winning team. The challenge behind this model is that:

- Many champions that if we perform one hot encoding, We will get several columns that would confuse our model and affect the accuracy in a bad way.
- The huge number of possible items will not have a meaning if we did one hot encoding for all of those.
- Maybe some champions and items combinations are more likely to win, but the winning process depends more on the tactics and strategies of playing.

7.2 Data Preprocessing

We have in our streamed data Champion names, Winning teams, Losing teams, Used Items, Match Duration Time Bans teams, and meta teams.
The data of interest are Champion names, Winning teams, Losing teams, Used Items, and Match Duration Time. But the data contains Champion and its items are combined in one column as in the below figure. This each cell in this column is [Champion ID, Item 1, Item 2, Item 3, Item 4, Item 5, Item 6]



7.3 Modifying data

- Giving Labels for the data by adding "Winner" column that is (1) if the first team is the winner and (0) if the second team is the winner. Actually, the first team is always the winning team. So, we split the data into two halves. One-half would stay the same and take a value (1) in the "Winner" column. The other one would replace team 1 with 2 and replace team 2 with 1 and takes a value (0) in the "Winner" column.
- splitting combined columns Like in the above figure into columns. For example, This figure would be split into (Champion ID, Item 1, Item 2, Item 3, Item 4, Item 5, Item 6)

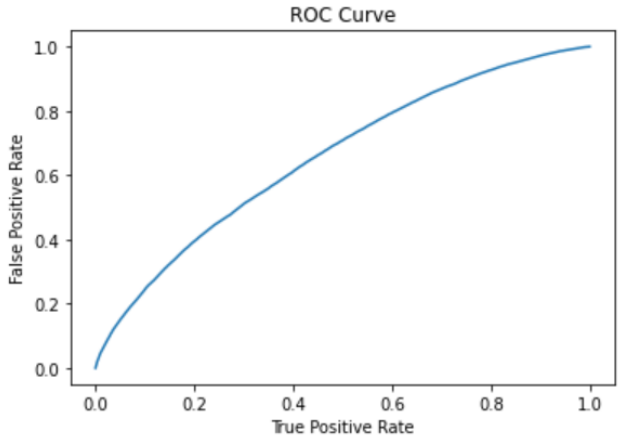
7.4 Pipeline

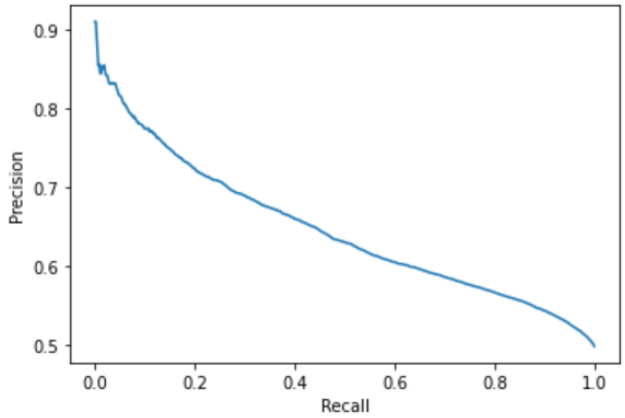
We used a Vector assembler to combine multiple input columns into a single output column of type vector. Then, we used fit and transform to deal with the data as a pipeline. After than, Assigning columns into a column of name features, and the "Winner" as label. So, the data is now in this form

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|    1|[1.0,0.0,3152.0,3...|
|    1|[1.0,1056.0,6653....|
|    1|[1.0,1056.0,6653....|
|    1|[1.0,242.0,166.0,...|
|    1|[1.0,302.0,10.0,5...|
|    1|[1.0,302.0,31.0,5...|
|    1|[1.0,302.0,66.0,5...|
|    1|[1.0,311.0,131.0,...|
|    1|[1.0,311.0,170.0,...|
|    1|[1.0,311.0,631.0,...|
+-----+--------------------+
```

Now, we trained our data on several models like:

- FM Classifier. (Accuracy 53.85%)
- Support Vector Classifier. (Accuracy 60.30%)
- One Vs All. (Accuracy 60.53%)
- Naive Bayes. (Accuracy 60.53%)
- Logistic Regression. (Accuracy 60.53%)

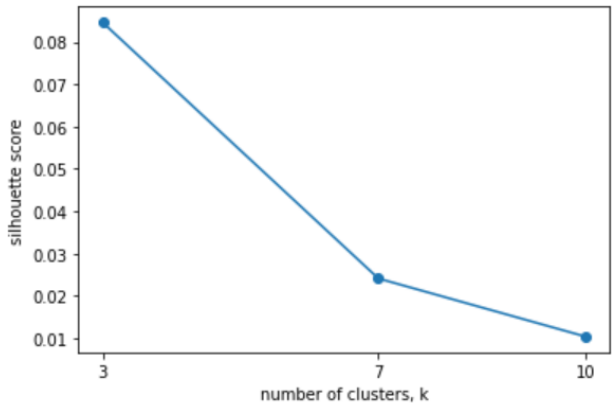




- Gradient-Boosted Tree Classifier. (Accuracy 62.29%)
- Random Forest. (Accuracy 60.54%)

7.5 Clustering for Both Champion and Items

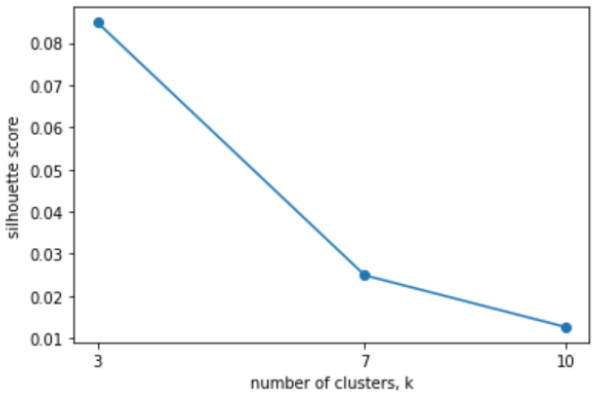
We applied K-Mean which measures the variation among the data points within each cluster.



It is so obvious that the silhouette score is not that good enough and that makes sense as the variation among data points is so high. For better performance, we will be in need of huge data.

7.6 Clustering for Items Only

We applied K-Mean on Items only.



The same as in the previous clustering. The silhouette score is not that good enough and that makes sense as the variation among data points is so high. For better performance, we will be in need of huge data.